CSN08x14

Scripting for Cybersecurity and Networks

Lecture 9:

Networking with Python

In this lecture

- *Sockets: network communication*
  - Connecting two machines / processes
- *Threading*
  - (just a mention)
- *Network traffic analysis*
  - Wireshark / pcap files overview
- *More about hashes: File content hashing*
- *Encoding*
- *File type analysis*
- *Other concepts*
  - Getting info about modules, methods etc.

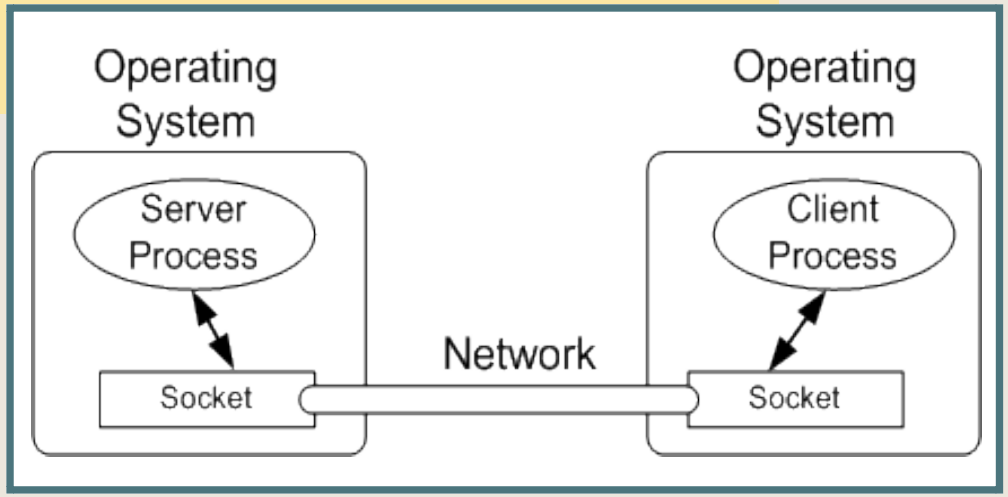
Go to www.menti.com

code **34 89 26**

We will discuss the coursework at the end of this lecture (separate slides)

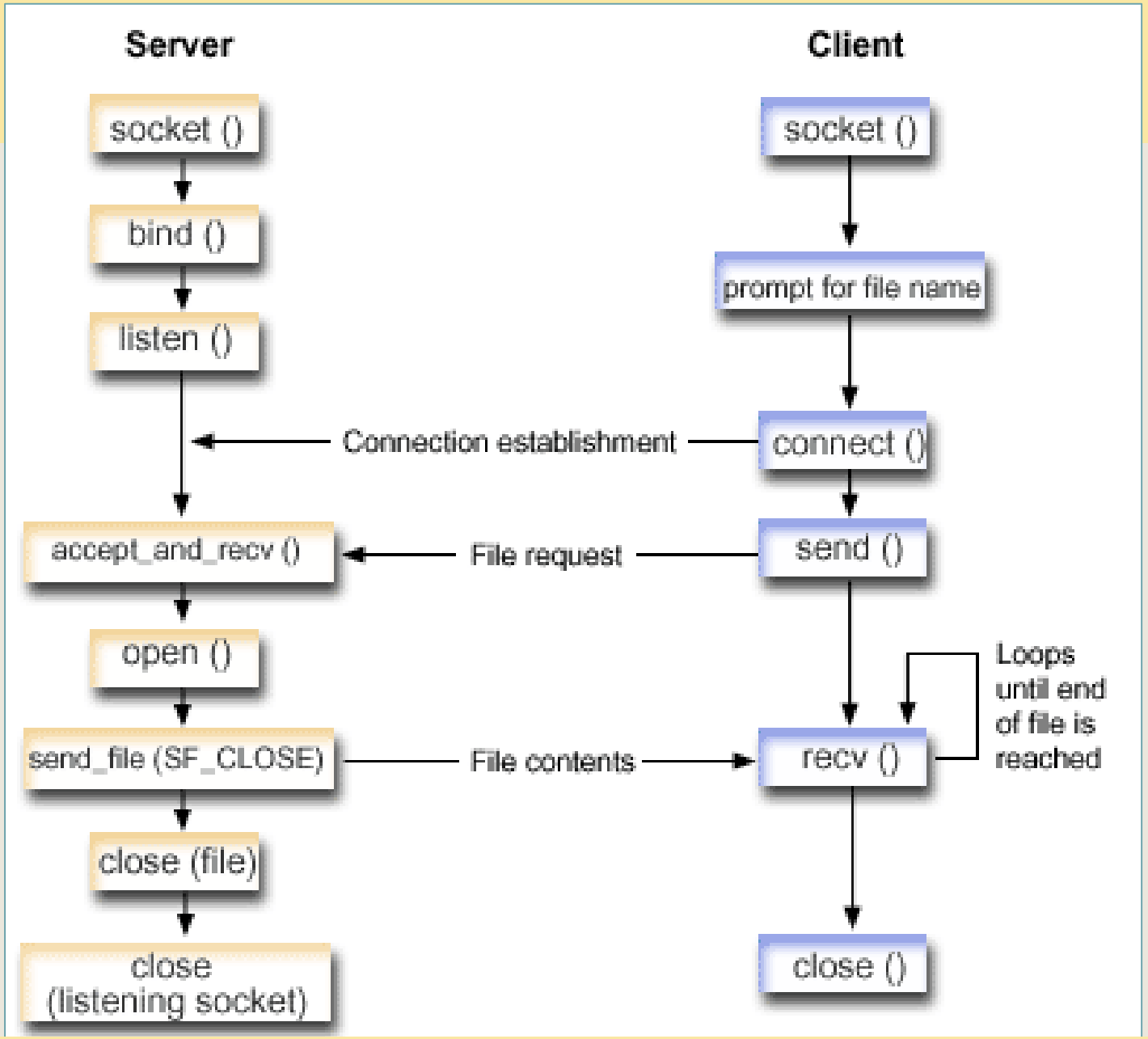
Sockets with Python

What are sockets?



- virtual endpoints of a communication channel
  - *between 2 programs or processes*
  - *on the same or different machines*
- the fundamental concept behind network applications
- enable network communication
- E.g. when you open google.com in your browser, your browser creates a socket and connects to google.com server.
  A socket on the google.com server accepts the connection and sends your browser the webpage that you see

Basic process diagram



http://www.ibm.com/support/knowledgecenter/ssw_i5_54/rzab6/rzab6500.gif

socket module in Python

`socket` module – low level module to create and manage sockets through Python

```
>>> import socket
>>> dir(socket)
['AF_APPLETALK', 'AF_DECnet', 'AF_INET', 'AF_INET6', 'AF_IPX', 'AF_IRDA', 'AF_SNA', 'AF_UNSPEC', 'AI_ADDRCONFIG', 'AI_ALL', 'AI_CAN
ONNAME', 'AI_NUMERICHOST', 'AI_NUMERICSERV', 'AI_PASSIVE', 'AI_V4MAPPED', 'AddressFamily', 'AddressInfo', 'CAPI', 'EAGAIN', 'EAI_AG
AIN', 'EAI_BADFLAGS', 'EAI_FAIL', 'EAI_FAMILY', 'EAI_MEMORY', 'EAI_NODATA', 'EAI_NONAME', 'EAI_SERVICE', 'EAI_SOCKTYPE', 'EBADF', '
EWOULDBLOCK', 'INADDR_ALLHOSTS_GROUP', 'INADDR_ANY', 'INADDR_BROADCAST', 'INADDR_LOOPBACK', 'INADDR_MAX_LOCAL_GROUP', 'INADDR_NONE'
, 'INADDR_UNSPEC_GROUP', 'IPPORT_RESERVED', 'IPPORT_USERRESERVED', 'IPPROTO_ICMP', 'IPPROTO_IP', 'IPPROTO_RAW', 'IPPROTO_TCP', 'IPP
ROTO_UDP', 'IPV6_CHECKSUM', 'IPV6_DONTFRAG', 'IPV6_HOPLIMIT', 'IPV6_HOPOPTS', 'IPV6_JOIN_GROUP', 'IPV6_LEAVE_GROUP', 'IPV6_MULTICAS
T_HOPS', 'IPV6_MULTICAST_IF', 'IPV6_MULTICAST_LOOP', 'IPV6_PKTINFO', 'IPV6_RECVRTHDR', 'IPV6_RECVTCLASS', 'IPV6_RTHDR', 'IPV6_TCLAS
S', 'IPV6_UNICAST_HOPS', 'IPV6_V6ONLY', 'IP_ADD_MEMBERSHIP', 'IP_DROP_MEMBERSHIP', 'IP_HDRINCL', 'IP_MULTICAST_IF', 'IP_MULTICAST_L
OOP', 'IP_MULTICAST_TTL', 'IP_OPTIONS', 'IP_RECVDSTADDR', 'IP_TOS', 'IP_TTL', 'IntEnum', 'IntFlag', 'MSG_BCAST', 'MSG_CTRUNC', 'MSG
_DONTROUTE', 'MSG_MCAST', 'MSG_OOB', 'MSG_PEEK', 'MSG_TRUNC', 'MSG_WAITALL', 'MsgFlag', 'NI_DGRAM', 'NI_MAXHOST', 'NI_MAXSERV', 'NI
_NAMEREQD', 'NI_NOFQDN', 'NI_NUMERICHOST', 'NI_NUMERICSERV', 'RCVALL_MAX', 'RCVALL_OFF', 'RCVALL_ON', 'RCVALL_SOCKETLEVELONLY', 'SH
UT_RD', 'SHUT_RDWR', 'SHUT_WR', 'SIO_KEEPALIVE_VALS', 'SIO_LOOPBACK_FAST_PATH', 'SIO_RCVALL', 'SOCK_DGRAM', 'SOCK_RAW', 'SOCK_RDM',
 'SOCK_SEQPACKET', 'SOCK_STREAM', 'SOL_IP', 'SOL_SOCKET', 'SOL_TCP', 'SOL_UDP', 'SOMAXCONN', 'SO_ACCEPTCONN', 'SO_BROADCAST', 'SO_D
EBUG', 'SO_DONTROUTE', 'SO_ERROR', 'SO_EXCLUSIVEADDRUSE', 'SO_KEEPALIVE', 'SO_LINGER', 'SO_OOBINLINE', 'SO_RCVBUF', 'SO_RCVLOWAT',
'SO_RCVTIMEO', 'SO_REUSEADDR', 'SO_SNDBUF', 'SO_SNDLOWAT', 'SO_SNDTIMEO', 'SO_TYPE', 'SO_USELOOPBACK', 'SocketIO', 'SocketKind', 'S
ocketType', 'TCP_MAXSEG', 'TCP_NODELAY', '_GLOBAL_DEFAULT_TIMEOUT', '_GiveupOnSendfile', '_LOCALHOST', '_LOCALHOST_V6', '__all__',
'__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', '_blocking_errnos', '_int
enum_converter', '_realsocket', '_socket', 'create_connection', 'dup', 'errno', 'error', 'errorTab', 'fromfd', 'fromshare', 'gaierr
or', 'getaddrinfo', 'getdefaulttimeout', 'getfqdn', 'gethostbyaddr', 'gethostbyname', 'gethostbyname_ex', 'gethostname', 'getnamein
fo', 'getprotobyname', 'getservbyname', 'getservbyport', 'has_ipv6', 'herror', 'htonl', 'htons', 'inet_aton', 'inet_ntoa', 'inet_nt
op', 'inet_pton', 'io', 'ntohl', 'ntohs', 'os', 'selectors', 'setdefaulttimeout', 'socket', 'socketpair', 'sys', 'timeout']
```

Socket object methods

- Methods for socket objects
  - *To view list, create socket s then use s.<tab>*
  - *Or*

```
>>> print ([method for method in dir(s)
            if callable(getattr(s, method))])
```

```
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>> s.
```

accept
bind
close
connect
connect_ex
detach
dup
family
fileno
get_inheritable

```
>>> s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>>> print([method for method in dir(s) if callable(getattr(s,method))])
['__class__', '__del__', '__delattr__', '__dir__', '__enter__', '__eq__', '__exit__', '__format__', '__ge__
', '__getattribute__', '__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__
lt__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str_
_', '__subclasshook__', '_accept', '_check_sendfile_params', '_decref_socketios', '_real_close', '_sendfile
_use_send', '_sendfile_use_sendfile', 'accept', 'bind', 'close', 'connect', 'connect_ex', 'detach', 'dup',
'fileno', 'get_inheritable', 'getpeername', 'getsockname', 'getsockopt', 'gettimeout', 'ioctl', 'listen', '
makefile', 'recv', 'recv_into', 'recvfrom', 'recvfrom_into', 'send', 'sendall', 'sendfile', 'sendto', 'set_
inheritable', 'setblocking', 'setsockopt', 'settimeout', 'share', 'shutdown']
```

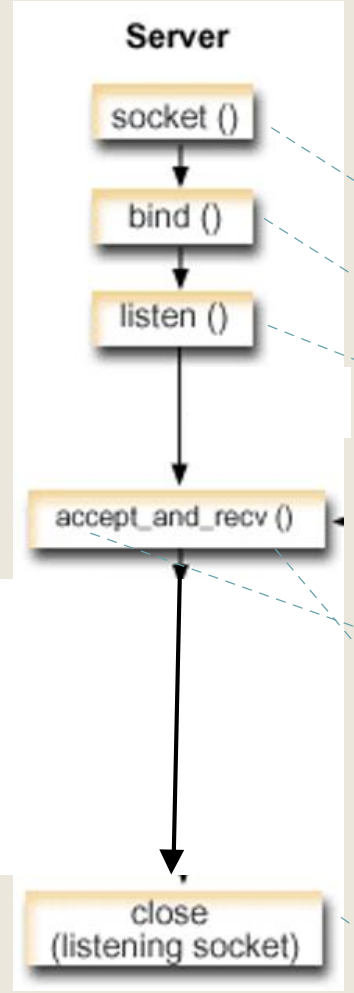
Methods for server highlighted red; methods for client highlighted green

Simple tcp socket communication in Python

What do we need?

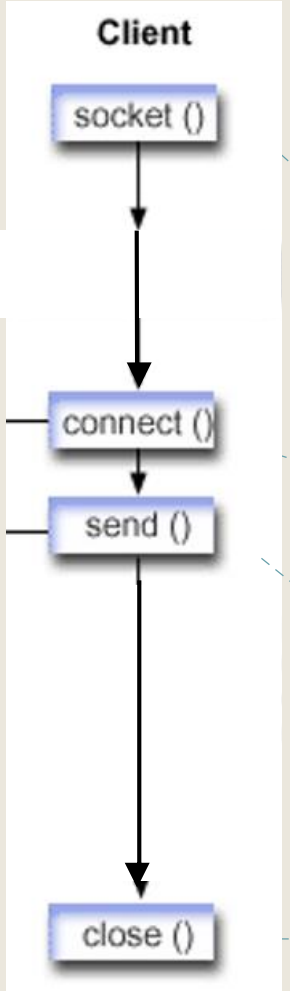
- Two scripts:
  one for server; one for client
  - *Both import socket module*
- Need address of server (IP address or similar)
  - *socket.gethostname() returns this*
- Need a port
  - *Can pick just about any port as long as it isn't blocked*

socket in Python: the server




```python
from socket import *

TCP_IP = '127.0.0.1' # use gethostname() or '127.0.0.1' for localhost
TCP_PORT = 5005 # The port to use.
BUFFER_SIZE = 1024 # Packet size - 1024 is standard.

s = socket(AF_INET, SOCK_STREAM) #Create connection object.
s.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1) #Set socket to be reusable to prevent
s.bind((TCP_IP, TCP_PORT)) # Bind...
s.listen(1) # ...and begin listening on specified ip address / port.

print (f'Server is running on {TCP_IP}:{TCP_PORT}')
print ('Awaiting connection...')

conn, addr = s.accept() #Awaiting an incoming connection.
print (addr, 'has connected.')
print ('Awaiting incoming message...')
while True:
    data = conn.recv(BUFFER_SIZE)
    if not data: break
    print ("Received message: ", data.decode('utf-8'))
    conn.sendall("Got that thanks!".encode('utf-8'))
conn.close()
```

socket in Python: the client




```
# client_start.py
# start script to create send message to a listening server socket
# Owen Lo Nov 2016

from socket import *

TCP_IP = 'petra-laptop' #The IP address of server to connect to.
TCP_PORT = 5005 # The port of server to connect to.
BUFFER_SIZE = 1024 #Packet size.

#your code here
pass # Your code to Create socket object.
pass # Your code to Connect to server.

print 'Connected to server %s:%s' % (TCP_IP, TCP_PORT)
print 'Enter a message to send to the server:'

message = raw_input()

pass # Your code to send message
pass # your code to close socket
```

Lab exercise for you to create the client!

demo

1. Run the server

```
 RESTART: E:\Petra\Dropbox\SET08115 Python\experiments and ideas\tcp_client_serv
er_messaging\tcp_client_server_messaging\server.py
Server is running on ME1C049-140375:5005
Awaiting connection...
```

2. Change hostname in client script and run in 2nd shell

```
 RESTART: E:\Petra\Dropbox\SET08115 Python\experiments and ideas\tcp_client_serv
er_messaging\tcp_client_server_messaging\client.py
Connected to server ME1C049-140375:5005
Enter a message to send to the server:
```

```
 RESTART: E:\Petra\Dropbox\SET08115 Python\experiments and ideas\tcp_client_serv
er_messaging\tcp_client_server_messaging\server.py
Server is running on ME1C049-140375:5005
Awaiting connection...
('146.176.164.91', 51975) has connected.
Awaiting incoming message...
```

3. Type message and enter to send

```
 RESTART: E:\Petra\Dropbox\SET08115 Python\experiments and ideas\tcp_client_serv
er_messaging\tcp_client_server_messaging\client.py
Connected to server ME1C049-140375:5005
Enter a message to send to the server:
Good morning!! :)
>>> |
```

```
 RESTART: E:\Petra\Dropbox\SET08115 Python\experiments and ideas\tcp_client_serv
er_messaging\tcp_client_server_messaging\server.py
Server is running on ME1C049-140375:5005
Awaiting connection...
('146.176.164.91', 51975) has connected.
Awaiting incoming message...
Received message: Good morning!! :)
>>>
```

Discussion

- scripts should also work on separate PCs
  (one PC can act as the client and the other as the server)
  so long as they have different IP addresses and the ports are not blocked.

- If anything unexpected happens
  (e.g. cannot receive message from client or server will not start due to exceptions) make sure server is not already running in background.

  To check, in Windows Command Prompt,
  type: `cmd > netstat -ano|findstr 5005`

- If that port (5005) is in use,
  kill it with: `cmd > tskill <process id>`
  and run server.py again.
  Alternatively, use another port if 5005 is taken.

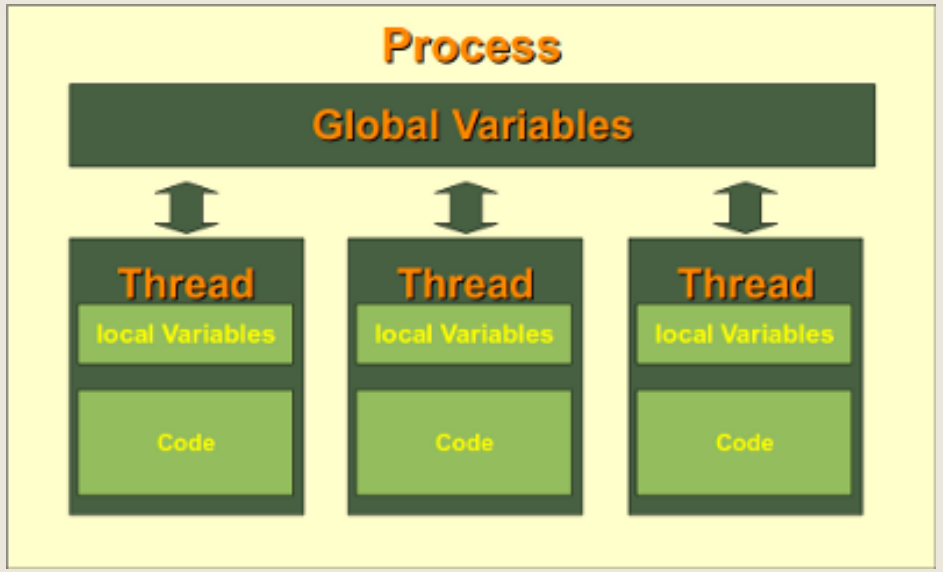
Discussion

- What is the purpose of the buffer size and the loop in the server?
  - *Demo with long test message*
  - *Why don't we need this in the client?*
- Sockets are usually single use:
  - *Will be discarded after one use and a new one opened*
- Socket types: TCP (connection-oriented), UDP (connectionless)
- To build a full messaging app would require more advanced concepts like multi-threading

Multi-threading in Python

Multi-threading

- Multiple threads allow concurrency and parallel execution of different actions
  - *Like running several processes at the same time, but:*
  - *Lightweight – less memory overhead*
  - *Share data space – easier communication with each other*
- `threading` module in Python



- Beyond the scope of this module (except for a very simple example)
- See https://www.tutorialspoint.com/python/python_multithreading.htm, http://www.python-course.eu/threads.php. (both written for Python 2 but still good)

Multi-threading: a very simple example

```python
# thread_example.py
# from http://www.saltycrane.com/blog/2008/09/simplistic-python-thread-example/

import time
from threading import Thread

def myfunc(i):
    print (f"sleeping 5 sec from thread {i}\n")
    time.sleep(5)
    print (f"finished sleeping from thread {i}\n")

for i in range(10):
    t = Thread(target=myfunc, args=(i,))
    t.start()
```

Q: What is the comma for?

Network Traffic Analysis

Analysing network traffic

- Very important for performance and computer security
- Real-time analysis required e.g.
  - *by Intrusion Prevention Systems (IPS)*
  - *For troubleshooting performance*
- Recorded traffic may be used for forensic investigations

Recording network traffic

- Several tools available
  - *Wireshark*
  - *Tcpdump*
  - *WinPcap*

- Wireshark
  - *Multi-platform packet analyser*
  - *Most widely used*
  - *Uses WinPcap behind the scenes for capture*

Remember:
You must have explicit permission to capture any traffic other than your own!!!

Wireshark

Filter

list of packets (frames)

More info about selected frame

Preview of contents (in hex viewer)

http.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

(http.request.method==GET or http.response==200) && tcp && http.content_type=="image/gif"

No.	Time	Source	Destination	Protocol	Length	Info
293	6.707355	23.21.54.131	146.176.164.91	HTTP	267	HTTP/1.1 200 OK (GIF89a) (GIF89a) (image/gif)
410	10.462669	66.117.29.34	146.176.164.91	HTTP	627	HTTP/1.1 200 OK (GIF89a) (GIF89a) (image/gif)
523	10.625860	23.21.54.131	146.176.164.91	HTTP	267	HTTP/1.1 200 OK (GIF89a) (GIF89a) (image/gif)
565	10.724975	216.58.201.2	146.176.164.91	HTTP	548	HTTP/1.1 200 OK (GIF89a) (GIF89a) (image/gif)
566	10.725589	216.58.201.2	146.176.164.91	HTTP	548	HTTP/1.1 200 OK (GIF89a) (GIF89a) (image/gif)
572	10.729160	216.58.212.98	146.176.164.91	HTTP	548	HTTP/1.1 200 OK (GIF89a) (GIF89a) (image/gif)
581	10.740683	216.58.212.98	146.176.164.91	HTTP	548	HTTP/1.1 200 OK (GIF89a) (GIF89a) (image/gif)
890	11.518109	138.108.96.100	146.176.164.91	HTTP	428	HTTP/1.1 200 OK (GIF89a) (GIF89a) (image/gif)
996	12.083670	66.117.29.34	146.176.164.91	HTTP	626	HTTP/1.1 200 OK (GIF89a) (GIF89a) (image/gif)
1111	12.464317	54.225.184.146	146.176.164.91	HTTP	267	HTTP/1.1 200 OK (GIF89a) (GIF89a) (image/gif)
2030	13.811735	54.225.184.146	146.176.164.91	HTTP	267	HTTP/1.1 200 OK (GIF89a) (GIF89a) (image/gif)
2154	14.311841	216.58.201.2	146.176.164.91	HTTP	548	HTTP/1.1 200 OK (GIF89a) (GIF89a) (image/gif)
2267	15.036207	138.108.96.100	146.176.164.91	HTTP	428	HTTP/1.1 200 OK (GIF89a) (GIF89a) (image/gif)
2359	15.677802	66.117.29.34	146.176.164.91	HTTP	626	HTTP/1.1 200 OK (GIF89a) (GIF89a) (image/gif)

```
> Frame 293: 267 bytes on wire (2136 bits), 267 bytes captured (2136 bits)
> Ethernet II, Src: CiscoInc_47:49:44 (a4:18:75:47:49:44), Dst: Pegatron_3b:eb:cd (e8:40:f2:3b:eb:cd)
> Internet Protocol Version 4, Src: 23.21.54.131, Dst: 146.176.164.91
> Transmission Control Protocol, Src Port: 80 (80), Dst Port: 53954 (53954), Seq: 1, Ack: 972, Len: 213
v Hypertext Transfer Protocol
  > HTTP/1.1 200 OK\r\n
    Cache-Control: no-cache, no-store, must-revalidate\r\n
    Content-Type: image/gif\r\n
    Expires: 0\r\n
    Pragma: no-cache\r\n
```

```
0000  e8 40 f2 3b eb cd a4 18  75 47 49 44 08 00 45 00   .@.;.... uGID..E.
0010  00 fd 5f 05 40 00 29 06  6d 52 17 15 36 83 92 b0   .._.@.). mR..6...
0020  a4 5b 00 50 d2 c2 20 67  d7 a1 32 72 14 aa 50 18   .[.P.. g ..2r..P.
0030  75 cf 96 d7 00 00 48 54  54 50 2f 31 2e 31 20 32   u.....HT TP/1.1 2
0040  30 30 20 4f 4b 0d 0a 43  61 63 68 65 2d 43 6f 6e   00 OK..C ache-Con
0050  74 72 6f 6c 3a 20 6e 6f  2d 63 61 63 68 65 2c 20   trol: no -cache,
0060  6e 6f 2d 73 74 6f 72 65  2c 20 6d 75 73 74 2d 72   no-store , must-r
0070  65 76 61 6c 69 64 61 74  65 0d 0a 43 6f 6e 74 65   evalidat e..Conte
0080  6e 74 2d 54 79 70 65 3a  20 69 6d 61 67 65 2f 67   nt-Type:  image/g
0090  69 66 0d 0a 45 78 70 69  72 65 73 3a 20 30 0d 0a   if..Expi res: 0..
```

Working with pcap files in Python

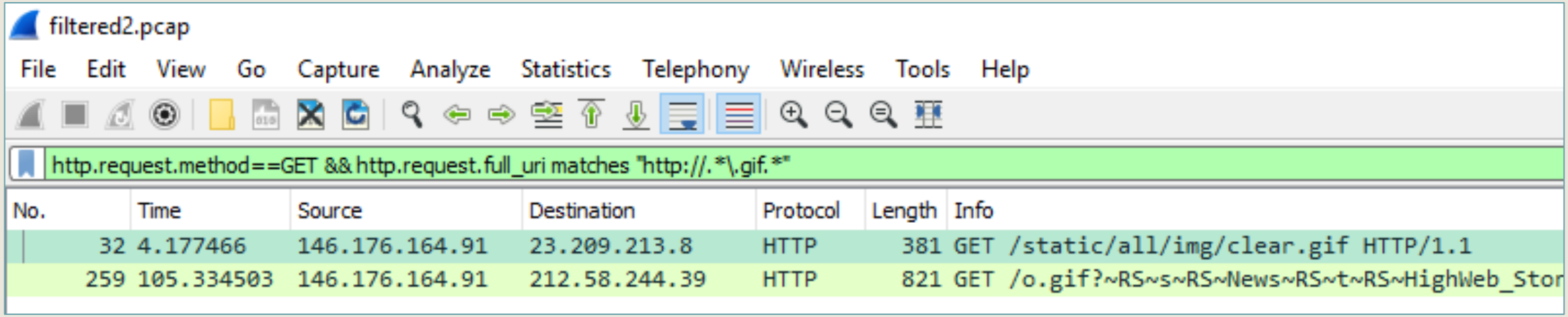
- **dpkt** module:
  for packet creation and parsing
  see https://jon.oberheide.org/blog/2008/10/15/dpkt-tutorial-2-parsing-a-pcap-file/
  (written for Python 2 but explains concepts well)

- **socket** module:
  provides socket operations and some related functions.
  Supports IP (Internet Protocol) sockets on all OS.
  Functions specific for a socket are available as methods of the socket object.

Example: extract downloaded .gif files from a packet capture

- any .gif displayed on a loaded webpage will be downloaded by the client
- In Wireshark, we could find these packets with the filter

```
http.request.method==GET
                && http.request.full_uri matches "http://.*\.gif.*"
```



Example: extract downloaded .gif files from a packet capture

- Python script to find "downloaded" gifs from a packet capture
- pcap_downloads.py provides this example, in working order

```
=========== RESTART: F:/Dropbox/CSN08114 Python/pcap_downloads.py ===========
[*] Analysing filtered2.pcap for gif files
[!] 146.176.164.91 downloaded /static/all/img/clear.gif from 23.209.213.8
[!] 146.176.164.91 downloaded /o.gif?~rs~s~rs~news~rs~t~rs~highweb_story~rs~i~rs
~37872899~rs~p~rs~99207~rs~a~rs~domestic~rs~u~rs~/news/uk-politics-37872899~rs~r
~rs~0~rs~q~rs~0~rs~z~rs~981~rs~ from 212.58.244.39
>>>
=========== RESTART: F:/Dropbox/CSN08114 Python/pcap_downloads.py ===========
[*] Analysing filtered3.pcap for gif files
No gif downloads found in this file
```

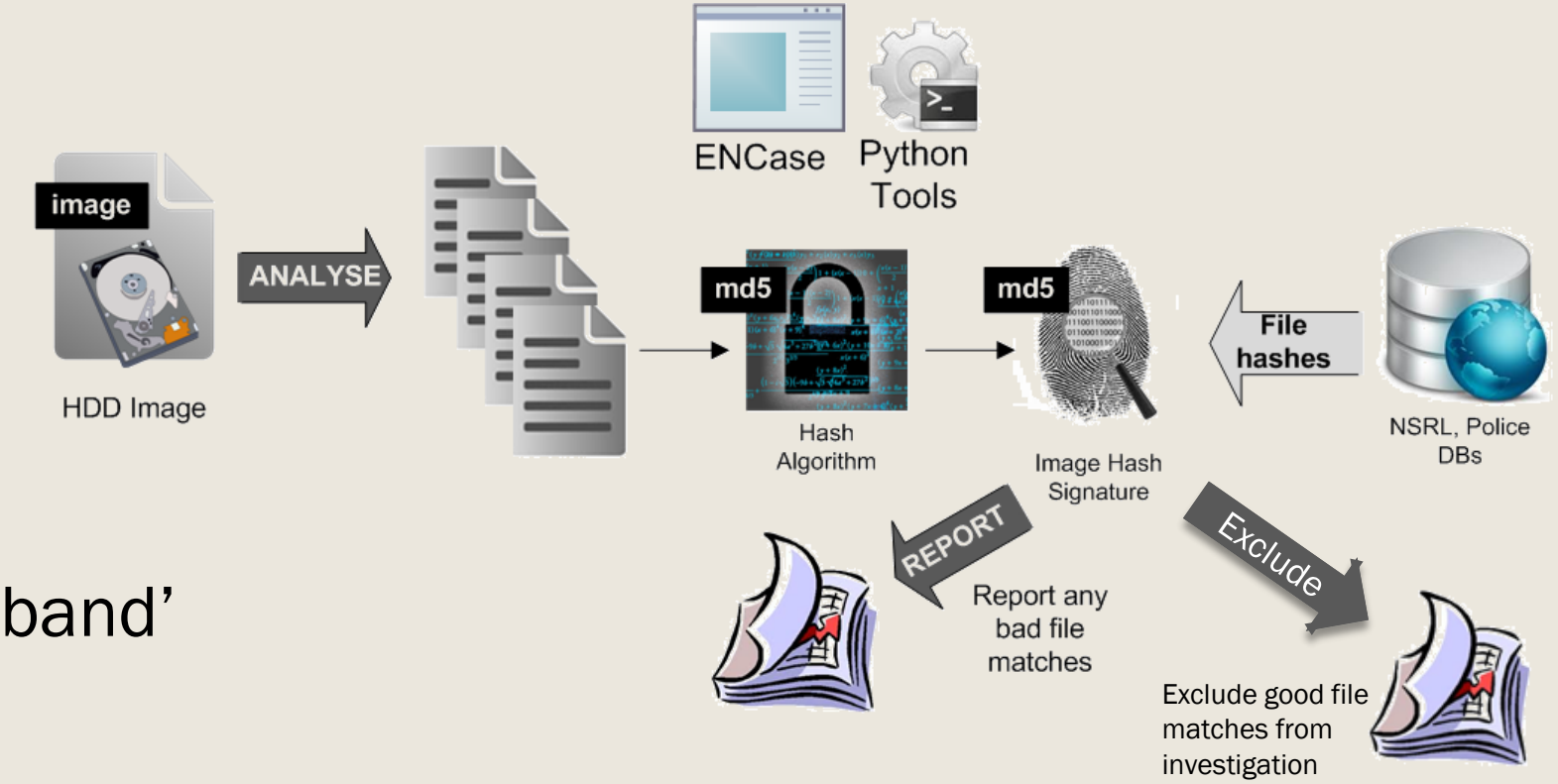
File content hashing

File content hashing

- So far, we have worked with hashes of short strings, such as passwords
- The entire contents of a file can also be hashed
  - *Check that download has not been tampered with*
  - *Check that a forensic image of a hard drive has not been tampered with during forensic analysis*
  - *Hash content of unknown files; compare to database of hashes*

File Hash Analysis: analysing unknown files

- Compare file hashes to hashes in Hash databases
- exclude known good files from investigation (whitelist)
- report any matches of known bad 'contraband' files (blacklist)



NSRL Good/bad hash database:
http://www.nsrl.nist.gov/index.html

File Hash Analysis with Python

File hash Analysis

- **Hash the file content** not the metadata

Mode
read, binary

```
>>> import hashlib
>>> f=open('c:\\temp\\a_file.txt','rb')
>>> file_content=f.read()
>>> md5_obj = hashlib.md5(file_content)
>>> print(md5_obj.hexdigest())
8d564f81a37edf4e8c591dd444983e59
```

'b' reads raw bytes rather than human formatted content

Q: Do you notice anything different than when we were hashing a string?

open(file, mode) : modes

- Mode is 2nd argument of open()
- Describes how file will be used
- Optional: default is 'r'
- Windows alters end-of-line character slightly when data is read or written. Corrupts binary data like JPEG or EXE. Use binary mode to prevent this
- When text files are read, the bytes in the file are decoded to human-readable strings. Use binary mode to prevent this

mode	
'r'	Read-only
'w'	Write-only (overwrites any existing file)
'a'	Append (add data to end)
'r+'	Read and write
'rb', 'wb', …	'b' appended to mode for binary files or to read raw bytes rather than strings

"with" to open files

- Good practice to use "with" to open files
- Closes file automatically at end of block

- `with open( 'filename', 'mode' ) as file_obj:`

```
try:
    with open('non-existent-file') as f:
        for line in f:
            print(line)

except IOError as err:
    print(f'!! IOError: {err}')
except Exception as err:
    print(f'!! Exception: {err}')
```

With code block deals with file, and automatically closes file

Digital Forensic File Hashes

```python
import sys, os
import hashlib

def get_hash(filename):
    """prints a hex hash signature of the file passed in as arg"""
    try:
        # Read File
        # *** your code ***

        # Generate Hash Signature
        # *** your code ***

        print (f'[+] get_hash() file: {filename} ', end='')
        #print (f'hash_sig: {file_hashsig}')

    except Exception as err:
        print (f'[-] {err}')

    finally:
        if 'f' in locals(): f.close()

def main():
        # Test case
        sys.argv.append(r'c:\temp\a_file.txt')
        # Check args
        if len(sys.argv) != 2:
                print ('[-] usage: file_hash filename')
                sys.exit(1)

        filename = sys.argv[1]
        get_hash(filename)

if __name__ == '__main__':
        main()
```

Generate File Hash Script – **file_hash.py**

Add your code here

Exception Handling
As we are dealing
With files

Digital Forensic File Hashes

```python
import sys, os
import hashlib

def get_hash(filename):
    """prints a hex hash signature of the file passed in as arg"""
    try:
        # Read File
        # *** your code ***

        # Generate Hash Signature
        # *** your code ***

        print (f'[+] get_hash() file: {filename} ', end='')
        #print (f'hash sig: {file_hashsig}')
```

```
===== RESTART: C:/Users/Petra/Dropbox/CSN08114 Pyt
hon/file_hash_start.py =====
[+] get_hash() file: c:\temp\a_file.txt
```

```python
        if 'f' in locals(): f.close()

def main():
        # Test case
        sys.argv.append(r'c:\temp\a_file.txt')
        # Check args
        if len(sys.argv) != 2:
                print ('[-] usage: file_hash filename')
                sys.exit(1)

        filename = sys.argv[1]
        get_hash(filename)

if __name__ == '__main__':
        main()
```

Create a test case so we can test in Python Shell – run as script

Add in a test case by adding an argument to sys.argv object

Scripting for Cybersec & Networks

Digital Forensic File Hashes

```python
import sys, os
import hashlib

def get_hash(filename):
    """prints a hex hash signature of the file passed in as arg"""
    try:
        # Read File
        # *** your code ***

        # Generate Hash Signature
        # *** your code ***

        print (f'[+] get_hash() file: {filename} ', end='')
        #print (f'hash_sig: {file_hashsig}')

    except Exception as err:
        print (f'[-] {err}')

    finally:
        if 'f' in locals(): f.close()

def main():
        # Test case
        sys.argv.append(r'c:\temp\a_file.txt')
        # Check args
        if len(sys.argv) != 2:
                print ('[-] usage: file_hash filename')
                sys.exit(1)

        filename = sys.argv[1]
        get_hash(filename)

if __name__ == '__main__':
        main()
```

Add code to get_hash() function to read file and generate hash

test the code

how?

Digital Forensic File Hashes

```python
import sys, os
import hashlib

def get_hash(filename):
    """prints a hex hash signature of the file passed in as arg"""
    try:
        # Read File
        # *** your code ***

        # Generate Hash Signature
        # *** your code ***

        print (f'[+] get_hash() file: {filename} ', end='')
```

```
======== RESTART: C:/Users/Petra/Dropbox/CSN08114 Python/file_hash.py ========
[+] get_hash() file: c:\temp\a_file.txt hash_sig: 8d564f81a37edf4e8c591dd444983e59
>>>
======== RESTART: C:/Users/Petra/Dropbox/CSN08114 Python/file_hash.py ========
[+] get_hash() file: c:\temp\b_file.txt hash_sig: 7ff56e77e4d0dc8b3c3e638cd6cd68fc
```

```python
def main():
        # Test case
        sys.argv.append(r'c:\temp\a_file.txt')
        # Check args
        if len(sys.argv) != 2:
                print ('[-] usage: file_hash filename')
                sys.exit(1)

        filename = sys.argv[1]
        get_hash(filename)

if __name__ == '__main__':
        main()
```

Generate File Hash Script – `file_hash.py`

Add hash file content code get_hash() function

Testing?
Test with different files to check different hash.
Test with 2 files with different name but same content to check hashes same

Check a file against a db of bad file hashes

- Create some "bad files", then get their hashes using script

- How to create a database of bad hashes?
  - *Need to store hash and filename*
  - *Needs to be quick to lookup as may be millions of bad hashes*
  - *mmmm hold on…*

Check a file against a db of bad file hashes

- Create database as a dictionary, containing hashes of bad files using our file_hash.py script

```
>>> import file_hash
>>> os.chdir('c:\\temp')
>>> bad_files = {}
>>> bad_files[file_hash.get_hash('badfile1.txt')]='badfile1.txt'
>>> bad_files[file_hash.get_hash('badfile2.txt')]='badfile2.txt'
>>> bad_files[file_hash.get_hash('badfile3.txt')]='badfile3.txt'
```

```
>>> print(bad_files)
{'8d564f81a37edf4e8c591dd444983e59': 'badfile1.txt', '7ff56e77e4d0dc8b3c3e638cd6cd68fc': 'badfile2.txt'
, '8a031a15a6c450f00f87d76b21e8dd98': 'badfile3.txt'}
```

- How do we then check files against our bad file hash lookup db?

Check a file against a db of bad file hashes

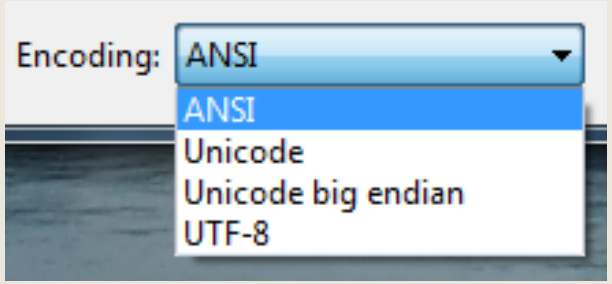
- Create hash of file under test,
  and match against bad hashes using an if statement:

```
>>> hash_to_test = file_hash.get_hash('a_file.txt')
>>> if hash_to_test in bad_files:
        print(f'!! Match found with {bad_files[hash_to_test]} ({hash_to_test})')

!! Match found with badfile1.txt (8d564f81a37edf4e8c591dd444983e59)
>>> hash_to_test = file_hash.get_hash('c_file.txt')
>>> if hash_to_test in bad_files:
        print(f'!! Match found with {bad_files[hash_to_test]} ({hash_to_test})')

>>>
```

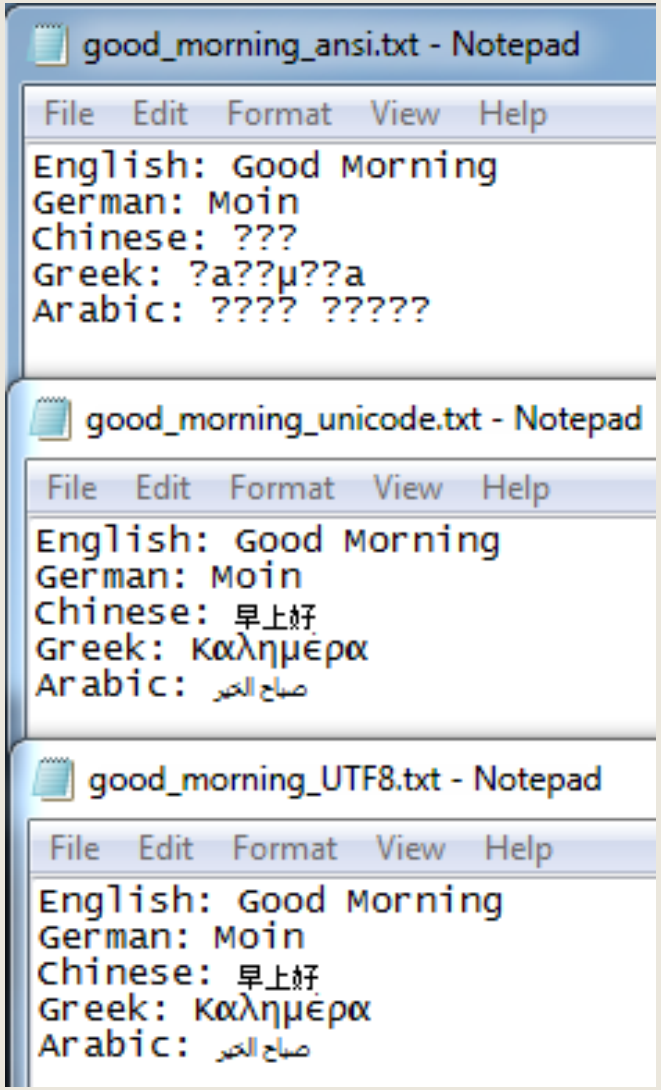
Tip: Normally we would have bad hash database stored as a file, and would read this in.

A Note on Encoding

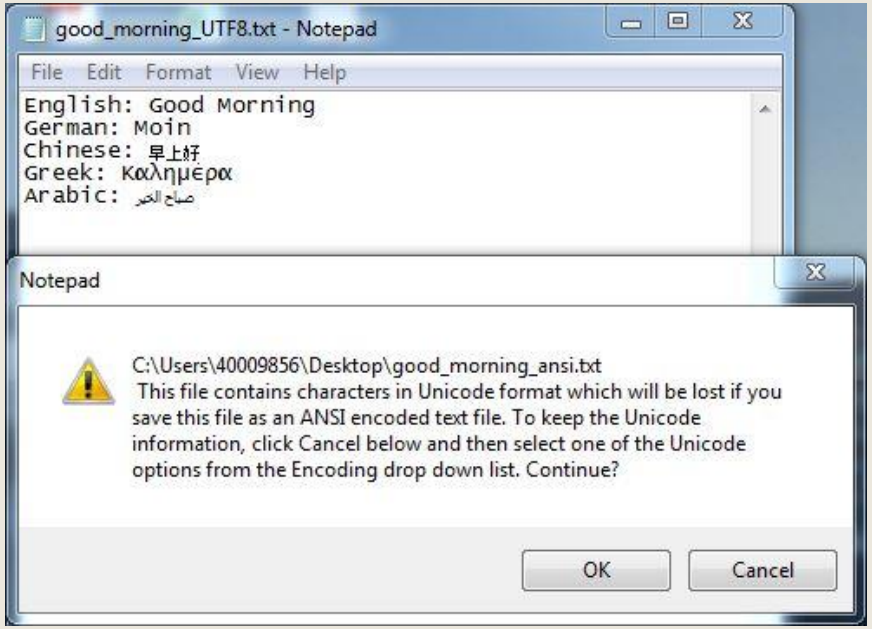
Dictionary saved 3 ways with Notepad



- ANSI (=ascii)
- UTF-8
- Unicode



- UTF-8 and Unicode look the same
- ANSI cannot save non-ascii characters



Behind the scenes: the bytes on the disk

- ANSI is short
  - *Starts with first letter*
  - *Non-ascii characters replaced with "?"*
- Unicode
  - *Starts with FF FE (indicates little-endian Unicode)*
  - *Every character uses 2 bytes*
- UTF-8
  - *Starts with EF BB BF (indicates UTF-8)*
  - *Characters use 1-4 bytes*

good_morning_unicode.txt | good_morning_UTF8.txt | good_morning_ansi.txt

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  45 6E 67 6C 69 73 68 3A 20 47 6F 6F 64 20 4D 6F  English: Good Mo
00000010  72 6E 69 6E 67 0D 0A 47 65 72 6D 61 6E 3A 20 4D  rning..German: M
00000020  6F 69 6E 0D 0A 43 68 69 6E 65 73 65 3A 20 3F 3F  oin..Chinese: ??
00000030  3F 0D 0A 47 72 65 65 6B 3A 20 3F 61 3F 3F B5 3F  ?..Greek: ?a??µ?
00000040  3F 61 0D 0A 41 72 61 62 69 63 3A 20 3F 3F 3F 3F  ?a..Arabic: ????
00000050  20 3F 3F 3F 3F 3F 0D 0A 0D 0A                     ?????....
```

good_morning_unicode.txt | good_morning_UTF8.txt | good_morning_ansi.txt

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  FF FE 45 00 6E 00 67 00 6C 00 69 00 73 00 68 00  ÿþE.n.g.l.i.s.h.
00000010  3A 00 20 00 47 00 6F 00 6F 00 64 00 20 00 4D 00  :. .G.o.o.d. .M.
00000020  6F 00 72 00 6E 00 69 00 6E 00 67 00 0D 00 0A 00  o.r.n.i.n.g.....
00000030  47 00 65 00 72 00 6D 00 61 00 6E 00 3A 00 20 00  G.e.r.m.a.n.:. .
00000040  4D 00 6F 00 69 00 6E 00 0D 00 0A 00 43 00 68 00  M.o.i.n.....C.h.
00000050  69 00 65 00 73 00 65 00 3A 00 20 00 E9 65         i.n.e.s.e.:. .ée
00000060  0A 00 47 00 72 00 65 00 65 00                    .N}Y....G.r.e.e.
00000070  6B 03 B1 03 BB 03 B7 03 BC 03                    k.:. .š.±.».·.¼.
00000080  AD 00 0A 00 41 00 72 00 61 00                    ..Á.±.....A.r.a.
00000090  62 00 20 00 35 06 28 06 27 06                    b.i.c.:. .5.(.'.
000000A0  2D 06 2E 06 4A 06 31 06 0D 00                    -. .'.D...J.1...
000000B0  0A                                               ......
```

Yellow: E
Green: new line
Red: 早 (zao)
Blue: λ

good_morning_unicode.txt | good_morning_UTF8.txt | good_morning_ansi.txt

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  EF BB BF 45 6E 67 6C 69 73 68 3A 20 47 6F 6F 64  ï»¿English: Good
00000010  20 4D 6F 72 6E 69 6E 67 0D 0A 47 65 72 6D 61 6E   Morning..German
00000020  3A 20 4D 6F 69 6E 0D 0A 43 68 69 6E 65 73 65 3A  : Moin..Chinese:
00000030  20 E6 97 A9 E4 B8 8A E5 A5 BD 0D 0A 47 72 65 65   æ©ä¸Šå¥½..Gree
00000040  6B 3A 20 CE 9A CE B1 CE BB CE B7 CE BC CE AD CF  k: ΚΚΙΙΛΙΗΜΙΝΙ
00000050  81 CE B1 0D 0A 41 72 61 62 69 63 3A 20 D8 B5 D8  .Ι±..Arabic: Øµ
00000060  A8 D8 A7 D8 AD 20 D8 A7 D9 84 D8 AE D9 8A D8 B1  ¨Ø§Ø. Ø§Ù„Ø®ÙŠØ±
00000070  0D 0A 0D 0A                                      ....
```

Why does it matter to Python?

- When a text file is read into a string, the bytes stored in it must be rendered as text strings
- This is decoding
- And only works if we decode with the same method as the file was encoded when it was written

- In Python, we can specify decoding when reading files and encoding when writing
  - *E.g.* f=open('cafe.txt', 'r', **encoding='utf_8'**)
- Default is utf-8

Demo with good morning dictionary

Why does it matter for forensics?

- Two files with exactly the same text content but encoded differently will have different hashes
- So question is, do we want to compare text content, or do we want to compare bytes?

Demo with good morning dictionary

File type Analysis

File type analysis

- In Windows, file extensions (.pdf, .jpg,…) are used to indicate the file type
- Very easy to change
  - *Circumvent some virus checkers*
  - *Anti-forensics*
- Almost all types of file have a file signature
  - *Also known as magic number*
  - *The first few bytes in the file itself*
  - *indicates the file type*
    - E.g. **FF D8** indicates jpg, **EF BB BF** indicates UTF-8 encoded text file
  - *Used by the Linux "file" command*
  - *Comprehensive lists available online e.g.* https://www.garykessler.net/library/file_sigs.html

File type analysis

- So when we download files from the web, we want to check if they are the sort of file they claim to be
- In lab - write a script to do this
- Contributes to coursework
- Method
  - *Read file signature*
  - *Compare to list of file signatures to determine the file type*
  - *Compare with file extension*

Practical

LAB 8 & LAB 9

- Because of the test last week, you have two labs for this week
- Both include some concepts and scripts that are useful for the coursework.
- Both include a number of optional exercises - skip these if you are short of time

Practical

LAB 9

- Network Packet Analysis

Network packet analysis

Your first task in the lab is to figure out how exactly the script pcap_downloads.py works!

Questions include:

- Where in the script are these filters applied?
  `http.request.method==GET`
  `&& http.request.full_uri matches "http://.*\.gif.*"`
  - *dpkt tutorial at https://jon.oberheide.org/blog/2008/10/15/dpkt-tutorial-2-parsing-a-pcap-file/ is a good starting point*
- What is the purpose of the **`socket.inet_ntoa`** function?
  - ***`>>> help(socket.inet_ntoa)`*** *should help with this*

Network packet analysis

Once you have understood the script, adapt it!

E.g.:

- Find downloaded jpg files
- Or use methods from the script to write a new script that will show traffic source and destination for each packet in the format

```
[+] Src: 54.194.240.68 --> Dst: 146.176.164.91
[+] Src: 54.69.185.4 --> Dst: 146.176.164.91
[+] Src: 54.69.185.4 --> Dst: 146.176.164.91
```

Network packet analysis

- Script to list all packets that are not from a specified source

```
 RESTART: C:\Users\Petra\Dropbox\SET08115 Python\experiments and ideas\geoip\pca
p_analysis.py
[*] analysing filtered2.pcap for packets not source 146.176.164.91
-------------------
[+] Src: 54.194.240.68 --> Dst: 146.176.164.91
[+] Src: 54.69.185.4 --> Dst: 146.176.164.91
[+] Src: 54.69.185.4 --> Dst: 146.176.164.91
[+] Src: 204.2.197.201 --> Dst: 146.176.164.91
[+] Src: 54.69.185.4 --> Dst: 146.176.164.91
[+] Src: 151.101.16.233 --> Dst: 146.176.164.91
[+] Src: 52.1.64.28 --> Dst: 146.176.164.91
[+] Src: 54.210.45.182 --> Dst: 146.176.164.91
[+] Src: 54.86.76.22 --> Dst: 146.176.164.91
[+] Src: 54.163.85.105 --> Dst: 146.176.164.91
[+] Src: 212.58.244.68 --> Dst: 146.176.164.91
[+] Src: 77.72.112.213 --> Dst: 146.176.164.91
[+] Src: 212.58.244.68 --> Dst: 146.176.164.91
```

Network packet analysis output formatting

- We want output like this:

```
146.176.164.91:54333 -> 212.58.246.109:80
146.176.164.91:54289 -> 23.209.210.242:80
146.176.164.91:54304 -> 23.209.210.242:80
146.176.164.91:54289 -> 23.209.210.242:80
```

- But we get this:

```
Connection details: b'\x92\xb0\xa4[':54333 -> b'\xd4:\xf6m':80
Connection details: b'\x92\xb0\xa4[':54289 -> b'\x17\xd1\xd2\xf2':80
Connection details: b'\x92\xb0\xa4[':54304 -> b'\x17\xd1\xd2\xf2':80
Connection details: b'\x92\xb0\xa4[':54289 -> b'\x17\xd1\xd2\xf2':80
```

- Need to convert b'\x92\xb0\xa4['
  to 146.176.164.91
- Can use socket.inet_ntoa() for this
  - *Or do it manually using shift and mask for this - see separate additional exercises file in moodle*